Software vulnerabilities and bug bounty programs

BY Carsten Bienz and Steffen Juranek

DISCUSSION PAPER

Institutt for foretaksøkonomi
Department of Business and Management Science

FOR 04/2020

ISSN: 1500-4066
May 2020

Software vulnerabilities and bug bounty programs

Carsten Bienz[a,1], Steffen Juranek[a,2]

[a] *Norwegian School of Economics (NHH)*

Abstract

Many software developers employ bug bounty programs that award a prize for the detection of bugs in their software. We analyze, in a model with asymmetric information, under which conditions a bug bounty program is beneficial for a software developer. In our model, a bug bounty program allows developers to perfectly discriminate between different types of bugs, and help to avoid reputation costs of exploited bugs. We find that the benefits of bounty program do not only depend on the characteristics of the underlying software but also that a bounty program crucially interacts with other elements of the security strategy.

This version: May 11, 2020

[1] Norwegian School of Economics (NHH), Department of Finance, Helleveien 30, 5045 Bergen, Norway, Phone: +47 5595 9374, Email: carsten.bienz@nhh.no

[2] Norwegian School of Economics (NHH), Department of Business and Management Science, Helleveien 30, 5045 Bergen, Norway, Phone: +47 5595 9863, Email: steffen.juranek@nhh.no

1. Introduction

In spring 2019 Apple introduced a bug bounty program that offers up to $1mn to anyone who is able to hack one of their products (Apple, 2019). This is part of their strategy to secure their software against intruders. Protection against being hacked gains more and more importance in an organizations' list of issues. Attacks on software systems constitute a significant threat to corporations, consumers and governments around the globe. For example, the WannaCry malware managed to infect more than 230 000 computers within a day and was estimated to have caused economic losses in excess of $4bn (Cooper, 2018).

The threat is especially pronounced for software developers. Their risk is highlighted by the observation that vulnerability announcements have a significantly negative effect on a software developer's market value (Telang and Wattal, 2007). The problem is further exemplified by the emergence of an active (black) market for zero-day vulnerabilities, i.e., software vulnerabilities that are unknown to, or at least unaddressed by the software developer. Prices on the market show an increasing willingness to pay for software vulnerabilities. For example, market prices for hacks that allow an attacker controlling any of Apple's iOS devices have risen to at least $1.5mn (Goodin, 2016).

The aim of our study is to analyze the role that bug bounty programs play in the overall protection strategy of software developers against exploits. In particular, we study under which circumstances it is beneficial for software developers to offer a bug bounty program, and how a bug bounty program interacts with other elements of the protection strategy.

In order to deal with the problem of cybersecurity, software developers traditionally answered technologically. First, they increase their efforts to improve the coding. Second, they employ "fuzzing" methods, i.e., they use random codes to detect anomalies of their programs. Finally, they answer with easier upgrades and increased security features. Microsoft, for example, has begun to force users of its Windows 10 operating system to upgrade in order to immunize their customer's systems once a security breach has been detected and resolved (Kelley, 2015).

However, technological mechanism are not sufficient to fully prevent hacks because "it is virtually impossible to design software that is free of vulnerabilities" (Choi et al., 2010, p. 869). Economic incentives can, therefore, be helpful to reduce the damage. One frequently encountered idea is to operate bug bounty programs and offer rewards for the disclosure of security vulnerabilities. Many software developers, not just Apple, offer indeed either monetary rewards themselves or cooperate with a service provider that administers a similar program.

Bug bounty programs appear to be a cost effective way to increase security. Finifter et al. (2013) analyze two existing vulnerability reward programs for the Firefox and Chrome browsers. They find that about 25% of the detected bugs stem from vulnerabilities disclosed via the program. The total amount of premium paid is by far below the wage costs of full time researchers.

Offering ex-ante prizes for the detection of vulnerabilities in a bug bounty program implies a commitment to pay. That solves an important problem of hackers when trying to sell a vulnerability to the software developer. Without disclosure the software developer cannot assess the bug. However, after disclosure the software developer may use the information to close the vulnerability without paying - a hacker may therefore refrain from offering the information to the software developer. Bug bounty programs introduce the legal basis for a commitment to pay for a pre-defined vulnerability. This commitment to pay allows the software developer to perfectly discriminate between type of vulnerabilities, and helps the software developer to avoid reputation costs of exploited bugs. We show that this effect is especially important the more likely and severe high value vulnerabilities are.

Furthermore, we show that the benefits of a bug bounty program crucially depend on the characteristics of the software and the possibilities of the vulnerability. A bug bounty program crucially interacts with other security activities such as an ex-post exploitation detection system and improved coding. Hence, it is crucial for software developers to follow an integrated approach that takes these interaction into account.

Within the small economic literature on cybersecurity, the patching strategy of software bugs has received particular attention (see, e.g., Arora et al., 2008; August and Tunca, 2011; August et al., 2019; Choi et al., 2010). This is because the optimal patching strategy is non-trivial because not all users update their software in a timely manner and a disclosure allows hackers to reverse-engineer the vulnerability. In other words, an update reveals the vulnerability and may lead to exploitation of non-updated versions.

Our analysis focuses on the discovery of the vulnerability itself rather than its technical resolution. Thereby, we abstract from the optimal patching strategy and the implementation of the patches. We build on the argument, that software security needs more than just a technological approach and that an economic approach can contribute to software security (Choi et al., 2010; August and Tunca, 2011).

2. Institutional background

2.1. General

Detecting a bug in a licensed software is in principle legal as long as it is not violating the End User License Agreement (EULA). However, accessing someone's computer without consent is almost always an illegal action.[3] Therefore, someone profiting financially from a vulnerability that allows others to access a third-party's computer operates at least in a grey zone. For this reason, the market for vulnerabilities is relatively opaque. In the following, we aim to shed some light into it with the limited information that are available.

There seems to exist an active market for exploits. Hackers are on the selling side of this market. A buyer can be anyone who benefits from the access to a vulnerability. That includes the developers of security software, the software developers themselves, prosecutors or intelligence services but also criminals. Intermediaries, such as brokers, operate between the sell and the buy sides. One example for a broker is Zerodium. Zerodium buys and sells information on vulnerabilities along with security recommendations

[3]In the US, for example, most forms of unauthorized access are considered as a crime under the Computer Fraud and Abuse Act.

to their clients. For example, Zerodium's predecessor VUPEN supplied the NSA with a vulnerability subscription service in 2012.[4]

Given the nature of the product being sold, market participants are particularly concerned about secrecy. However, the hack of the software security firm "Hacking Team" offers a unique opportunity to gain insight into the business practices on the market.[5] Hacking Team is a producer of software that allows the surveillance of and access to computer systems, for example, by criminal prosecutors and intelligence services. Hacking Team itself actively solicited the purchase of vulnerabilities and paid hackers for them. Ironically, this company was hacked itself in 2015, and 400GB of its internal data was released publicly.[6]

Part of this release are information on the arrangement between a hacker and Hacking Team. The hacker Vitaliy Toropov offered information about a vulnerability in Adobe Flash Player. The arrangement, fixed in an email, specifies the payments and its timing:[7]

> 1) The price is US$45,000.00 for the non-exclusive sale of (...) any special discount for the "first" deal together will be greatly appreciated :)
> 2) information about vulnerability in Adobe Flash Player 9.x/10.x/11.x with the RCE exploit for the current Flash Player 11.9.x for Windows 32/64-bit and OS X 64-bit. The exploit code executes custom payloads with the privileges of the target process (it doesn't give any privilege escalation or a sandbox escape).
> 3) I send you sources (today or on next Monday, on your choice). I guess our guys can test it starting from Tuesday 29th.
> 4) The first payment is $20,000.00 which should be done by you in October 2013 via bank wire transfer.
> 5) The second payment is $15,000.00 in November 2013.
> 6) The final payment is $10,000.00 in December 2013.
> 7) The payment process can be stopped by you in case if this 0day is patched by vendor.
> 8) You promise to not report this 0day to vendor or disclosure it before the patch. obviously it is not our interest!

[4]The website Muckrock was able to get hold of this contract via a Freedom of Information Act (FIOA) request. The document and the request are available here: `https://www.muckrock.com/foi/united-states-of-america-10/vupen-contracts-with-nsa-6593/#comms`.

[5]A nice non-technical write-up can be found here: `https://tsyrklevich.net/2015/07/22/hacking-team-0day-market/`.

[6]The emails related to this hack can be directly accessed on Wikileaks; `https://wikileaks.org/hackingteam/emails/`.

[7]`https://wikileaks.org/hackingteam/emails/emailid/62010`.

This exemplary arrangement shows that even though the contracting parties struggle with enforceability, the arrangement seems to be incentive compatible. Hacking Team had the option to test part of the exploits, and if the hack would turn out not to be as valuable as promised for Hacking Team, they could stop the payments. On the other hand, Hacking Team could not simply take the information without paying because then the hacker could punish the company by reporting the vulnerability to the vendor. Note that the same incentives are not present in a deal between a hacker and the developer of the underlying software, i.e., the vendor. In that case, the developer has the incentive to patch the software without paying. Hence, a hacker would only offer a hack to the vendor for an upfront payment, prohibiting a detailed analysis of the hack by the vendor before the transaction.

2.2. Bug bounty programs

Bug bounty programs are a relative recent innovation. Mozilla, the foundation behind the Firefox web-browser was among the first to implement one in 2005 (Finifter et al., 2013). In recent years, many major software developers followed and introduced bug-bounty programs. Apple was one of the last software developers to introduce a program that covers all of their products. We summarize the programs of some major software developers in Table 1.

Firm	**Product**	**Date**	**Monetary Reward**	**Max Amount**
Adobe		4/2015	no	NA
Apple	iOS	8/2016	yes	$200k
Apple	All	12/2019	yes	$1mn
Facebook		10/2011	yes	unknown
Google	Chrome	1/2010	yes	$1,337
Google	Android	9/2018	yes	$20k
Mozilla	Firefox	2004	yes	$500
Microsoft	Windows	6/2013	yes	$100k

This table shows adoption dates for major software developers bug bounty programs (BBP) or vulnerability disclosure programs (VDP), as announced by press reports. Maximum amounts are for the announcement date. In a BBP a monetary reward is paid whereas a VDP only hands out non-monetray rewards, whith Adobe an example for the latter approach.

Table 1: Adoption of selected vulnerability reward programs or bug bounty programs

Bug Bounty programs offer a prize to hackers, denoted as researchers in that context, for identifying vulnerabilities with a specified ability. The software developer commits to paying the prize, and the hackers can take legal actions if the software developer refrains from doing so. Hence, the bug bounty programs commit software developers not only to pay for a vulnerability, they also legally indemnify hackers by explicitly allowing hackers to test computer system, as can be seen from Apple's Terms and Conditions (Apple, 2019).

Almost all programs we looked at distinguish payouts according to their severity. Apple's current program, for example, starts with payouts from $25,000 but pays up to $1mn for the most severe one (Apple, 2019).

3. The model

We include particular features of this institutional background into a theoretical framework aiming to analyze the benefits of a bug bounty program. The main elements are the commitment problem of software developer, the market for exploits and search efforts by the software developer after an exploit is detected.

3.1. The set-up

We start by introducing the players, their actions and other security procedures implemented by the developers.

Players: A hacker (H) works on detecting security gaps in a software manufactured by a software developer (S). If the hacker detects a vulnerability, she can exploit the gap for up to two periods. There exist two types of security vulnerabilities, L-type and H-type vulnerabilities. Both types differ in the damage they cause per period for the software developer, i.e., $D_H > D_L$. The value of exploiting a vulnerability equals αD_H or αD_L per period, with $\alpha > 0$. Note that $\alpha \neq 1$ implies an asymmetry between the damage that is caused and the value that is created for an intruder. One can easily imagine that the exploitation of a vulnerability leads to a reputation damage of the software developer, implying that $\alpha < 1$. In contrast, $\alpha > 1$ resembles a case where the information that can

be acquired with the vulnerability by an intruder is much more valuable than the damage to the software developer. This can be for example vulnerabilities that are used by law enforcement agencies against potential terrorists.[8]

Actions: The hacker searches for security gaps, and finds a gap of type L with probability p and a gap of type H with probability $1-p$. The parameter p can be interpreted as the ex-ante security level of the software.

Once the hacker detects a security gap, she observes its type. The hacker can monetize the vulnerability by either exploiting it herself, by selling it on the market, or by offering it to the software developer for sale. However, in the latter case, the software developer does not observe the type because the hacker can not easily disclose the information. If she would do so, the software developer would receive all the necessary information to close the gap immediately.[9]

The software developer also employs an ex-post detection system. After a vulnerability is exploited, the software developer observes the type, and invests resources in trying to close it. Furthermore, the software developer can implement a bounty program at fixed costs F. With a bounty program the software developer defines different prizes for the two types of vulnerabilities, and it commits to pay this prize if the required criteria are met.[10]

Timing: If the software developer implements a bounty program, it defines prizes in $t = 0$. In $t = 1$, the hacker searches and finds a security gap. If the hacker accepts the respective prize, the game ends. Otherwise, the hacker approaches the software developer notifying her of having found a security gap in $t = 2$. Then, the software developer makes

[8] One example is the famous San Bernadino Case where the FBI wanted Apple to unlock an encrypted iPhone. Apple refused but the FBI eventually managed to access the phone with the help of another party. A summary is provided by Wikipedia: `https://en.wikipedia.org/wiki/FBIApple\_encryption\_dispute`

[9] In principle, this commitment problem may be solved by repeated interaction. However, because of the large number of individual hackers, there rarely exists repeated interaction between a software developer and a particular hacker. That argument also explains the emergence of intermediaries, such as "HackerOne", which provide an alternative solution to the commitment problem.

[10] In order to credibly commit, the software developer can predefine the ability of a vulnerability and cover this by a contract. Additionally, it can also outsource the decision authority to an external committee.

a take-it-or-leave-it offer B to the hacker in exchange for the information. If the hacker accepts the offer, the game ends. If she rejects the offer, she monetizes the gap by selling it on the market or exploiting it herself. The software developer incurs the damage D_i once.

In $t = 3$, the software developer invests in detection in order to avoid further damage. In particular, the software developer decides upon the detection probability q. This choice is costly as the software developer incurs costs of $\frac{1}{2k}q^2$. If the software developer is unable to detect the gap, it incurs an additional damage of D_i.

An exploitation of the gap leads to revenues for the hacker of D_i in $t = 2$, and the same amount in $t = 3$ if the software developer is unable to close the gap with its ex-post detection system. Figure 1 summarizes the timing.

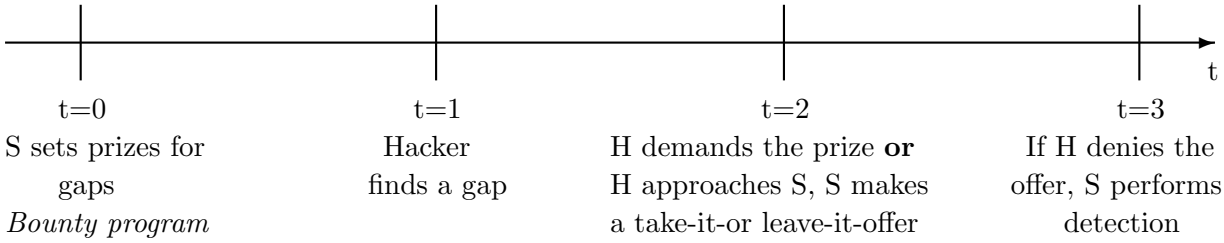

Figure 1: Timing

Our aim is to analyze the benefits of a bounty program. Therefore, we compare the equilibria with and without a bounty program, i.e., settings that start in period $t = 1$ and $t = 0$, respectively.

3.2. No bounty program

We start our analysis with the benchmark cases of no bounty program. In this case, the game starts in period $t = 1$. We solve the respective game by backward induction. In case of a failure of negotiations, the software developer observes the type of the gap and invests in period $t = 3$ in detection. The software developer maximizes its expected payoff:

$$\max_{q_i} \quad \Pi^{S,t=3} = -D_i(1 - q_i) - \frac{1}{2k}q_i^2 - D_i.$$

Consequently, the software developer invests $q_i^* = kD_i$, the software developer's expected payoff equals $\Pi^S = -D_i(2 - \frac{kD_i}{2})$, whereas the hacker expects a payoff of $\Pi^H = \alpha D_i(2 - kD_i)$.

Both players take this decision into account in period $t = 2$. The software developer makes the hacker an offer B. The hacker can either accept or reject the offer. If she rejects the offer, both proceed to stage 3. Because the software developer does not observe the type of vulnerability, it cannot discriminate with different offers. Therefore, the software developer has three options. First, the pooling option P, the software developer makes an offer that the hacker accepts for both types of vulnerabilities. The minimal offer that a hacker accepts for both types of vulnerabilities equals $\alpha D_H\,(2 - kD_H)$. Second, option L, the software developer makes an offer that only hackers with an L-type hacks accept. In this case, the offer equals $\alpha D_L\,(2 - kD_L)$. Third, option 0, the developer makes no offer. The developer's payoff for the three cases follow as (the subscript P stands for pooling of H- and L-type hacks at the same price).

$$\Pi_P^{S,t=2} = -\alpha D_H\,(2 - kD_H)\,, \tag{1}$$

$$\Pi_L^{S,t=2} = -(1-p)D_H\left(2 - k\frac{D_H}{2}\right) - p\alpha D_L\,(2 - kD_L)\,, \tag{2}$$

$$\Pi_0^{S,t=2} = -(1-p)D_H\left(2 - k\frac{D_H}{2}\right) - pD_L\left(2 - k\frac{D_L}{2}\right). \tag{3}$$

It becomes obvious that the optimal offer depends on the different parameters. Therefore, we define critical levels of α for which the software developer is indifferent between the three options. Parameter α denotes the market value of the vulnerability relative to the damage of the developer. Hence, the smaller α, the more it pays off for the developer to pay for a hack rather than accepting exploitation. However, if α is high, the developer may not be willing to pay for the hack because the demanded price will be higher than the expected damage.

We denote the critical α for which the software developer is indifferent between options P and L by α_{LP}, between option P and 0 by α_{0P}, and between option L and 0 by α_{0L}.

Solving for these critical values gives

$$\alpha_{LP} = \frac{(1-p)D_H\left(2-k\frac{D_H}{2}\right)}{D_H(2-kD_H)-pD_L(2-kD_L)}, \tag{4}$$

$$\alpha_{0P} = \frac{(1-p)D_H\left(2-k\frac{D_H}{2}\right)+pD_L\left(2-k\frac{D_L}{2}\right)}{D_H(2-kD_H)}, \tag{5}$$

$$\alpha_{0L} = \frac{2-k\frac{D_L}{2}}{2-kD_L} = 1+\frac{k\frac{D_L}{2}}{2-kD_L} > 1. \tag{6}$$

Whereas paying off both types is only optimal if $\alpha < \alpha_{LP}$ and $\alpha < \alpha_{0P}$, paying only for L-type hacks requires $\alpha > \alpha_{LP}$ and $\alpha < \alpha_{0L}$. Furthermore, not paying at all can only be optimal if $\alpha > \alpha_{0L}$ and $\alpha > \alpha_{0P}$.

Note that α_{0L} is larger than one. The intuition behind this observation is that the developer is in principle willing to pay the amount that equals the damage he would incur otherwise. However, the developer also has to bear the ex-post detection costs. Whereas the ex-post detection system affects the price that the hacker demands via the detection probability, the realized detection costs do not. This increases the developer's willingness to pay for the hack; for $k = 0$, it holds that $\alpha_{0L} = 1$.

In order to characterize the optimal offer, the order of the critical values matter. Comparing Eq. (4) to (6) allows us to state Lemma 1.

Lemma 1.

1. *For* $p > \tilde{p} \equiv 1 - \alpha_{0L}\frac{\Delta_1}{\Delta_2}$*, it holds that* $\alpha_{LP} < \alpha_{0P} < \alpha_{0L}$*,*
   with $\Delta_1 \equiv D_H(2-kD_H) - D_L(2-kD_L)$ *and* $\Delta_2 \equiv D_H\left(2-k\frac{D_H}{2}\right) - D_L\left(2-k\frac{D_L}{2}\right)$
2. *For* $p \leq \tilde{p}$*, it holds that* $\alpha_{LP} \geq \alpha_{0P} \geq \alpha_{0L}$

Proof. See the Appendix.

The consequences of Lemma 1 can be best illustrated by Figures 2 and 3. If $p > \tilde{p}$ there exists a parameter region for which all three candidate offers can be optimal. If α is small, it is optimal to pay off both types of hacks at the high price. This strategy becomes more costly with a higher α, and eventually, it becomes optimal to discriminate between the two types. Because the developer does not observe the type, she can only rely on self-selection and pay only for L-type hacks. The more α increases the more these

payments increase, and at one point the developer prefers not paying and to incur the damage.

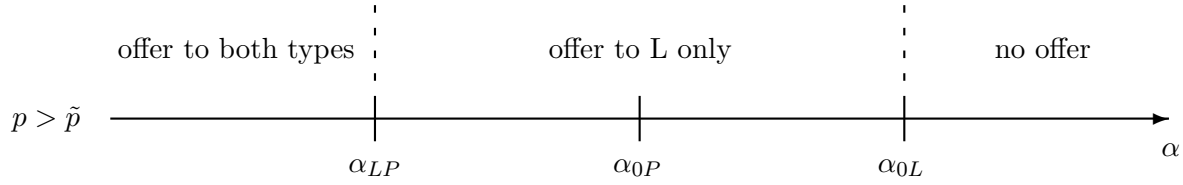

Figure 2: Optimal offers for $p > \tilde{p}$

If p is sufficiently low, i.e., if $p \leq \tilde{p}$, the high type hacks are relatively dominant among the hacks. In this case, the middle case of paying of only L-type gaps vanishes, and the developer pays off both types of hacks for small α or pays for none for high α. This is because by moving from the P-case to the L-case, the software developer profits from separating the two types but loses on the H-types if α is relatively large. Hence, if the share of H-types is sufficiently high, moving from P to L is not profitable. For the limit case of $p = \tilde{p}$, all critical values of α are equal to each other. We consider the case $p \leq \tilde{p}$ to be the less interesting case because it implies that the share of H-type hacks is so high that the software developer is willing to never differentiate between the two types.

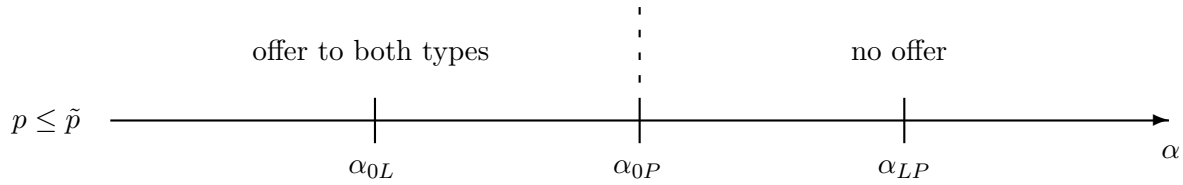

Figure 3: Optimal offers for $p \leq \tilde{p}$

This reasoning allows us to derive the equilibrium if the software developer does not employ a bounty program:

Lemma 2.

1. *For $p > \tilde{p}$ the software developer*
   (a) *pays an uniform price for both types of hacks if $\alpha \leq \alpha_{LP}$,*
   (b) *pays only for L-type hacks if $\alpha_{L,P} < \alpha \leq \alpha_{0L}$,*
   (c) *pays for none of the hacks if $\alpha > \alpha_{0L}$.*
2. *For $p \leq \tilde{p}$, the software developer*
   (a) *pays an uniform price for both types of hacks if $\alpha \leq \alpha_{0P}$,*
   (b) *pays for none of the hacks if $\alpha > \alpha_{0P}$.*

3.3. Bounty program

For simplicity, we concentrate from now on only on the more relevant case $p > \tilde{p}$. With a bounty program the games starts already in $t = 0$. In the bounty program, the developer commits to pay a pre-defined prize for the two types, P_H and P_L. After finding a gap in $t = 1$, the hacker can demand the respective prize. In case the hacker rejects the prize, she may approach the developer again in $t = 2$, and we are back in the game presented in section 3.2.

In essence, the bounty program allows the software developer to discriminate between the hackers at the fixed costs F. Therefore, the software developer has in principle four options for the design of the prizes of the bounty program. First, option S she defines separate prizes that will be accepted by both types of hacks. Second, option H, she defines a prize that will be accepted by H-type hacks only. Third, option L, she defines a prize that will be accepted by L-type hacks only. Fourth, option 0, she defines no prize.

For the separation option S the prizes have to be designed such that hackers with both types of hacks will accept it. The lowest possible prizes that ensure this are $P_H = \alpha D_H (2 - kD_H)$ for H-type hacks and $P_L = \alpha D_L (2 - kD_L)$ for the L-type hacks. The prizes ensure that a hacker with an H-type vulnerability has no incentive to reject the prize as she will not be able to gain more in a later stage. Neither has a hacker with an L-type hack an incentive to reject the offer. If the hacker would do so, the software developer can identify the hack right away because it knows that all hackers with a H-type hack accepted the prize. Consequently, it would also offer P_L in stage 2, which the hacker would accept. Hence, hackers with a L-type hack cannot improve. We denote the profit in this alternative as $\Pi_S^{S,t=0}$.

The prize for option H follows as $P_H = \alpha D_H (2 - kD_H)$. A hacker with a H-type hack would never accept a lower payment. As there are only L-type hackers left in $t = 2$, this case is only different from the first if the software developer has no incentive to buy the vulnerability in $t = 2$.

Unfortunately, option L is not a viable one. Defining a prize of $P_L < \alpha D_H (2 - kD_H)$

cannot be part of a pure strategy equilibrium as it would not be accepted by L-type hackers. This is because hackers with L-type hacks profit from being pooled with H-type hacks in $t = 2$. Hence, this case as well as option 0, i.e., offering no prize, leads to the result described in Lemma 2.

Hence, we have to compare profits of the software developer offering prizes for both with offering prizes for only H-type hacks with offering no prize.The profits of the three alternatives are given by[11]

$$\Pi_S^{S,t=0} = -(1-p)\,\alpha D_H\,(2-kD_H) - p\alpha D_L\,(2-kD_L)\,, \tag{7}$$

$$\Pi_H^{S,t=0} = -(1-p)\,\alpha D_H\,(2-kD_H) - pD_L\left(2-k\frac{D_L}{2}\right), \tag{8}$$

$$\Pi_0^{S,t=0} = \begin{cases} \Pi_P^{S,t=2} = -\alpha D_H\,(2-kD_H) & \text{if } \alpha \leq \alpha_{LP} \\ \Pi_L^{S,t=2} = -(1-p)\,D_H\left(2-k\frac{D_H}{2}\right) - p\alpha D_L\,(2-kD_L) & \text{if } \alpha_{LP} < \alpha \leq \alpha_{0L} \\ \Pi_0^{S,t=2} = -(1-p)\,D_H\left(2-k\frac{D_H}{2}\right) - pD_L\left(2-k\frac{D_L}{2}\right) & \text{if } \alpha > \alpha_{0L} \end{cases} \tag{9}$$

It is easy to see that $\Pi_S^{S,t=0}$ dominates $\Pi_P^{S,t=2}$. Hence, we only have to compare $\Pi_S^{S,t=0}$, $\Pi_H^{S,t=0}$, $\Pi_L^{S,t=2}$ and $\Pi_0^{S,t=2}$. For this comparison we derive the critical values of α for which the software developer is indifferent between the pairs of options:

$$\Pi_S^{S,t=2} > \Pi_H^{S,t=0} \text{ if } \alpha < \alpha_{HS} = \frac{2-k\frac{D_L}{2}}{2-kD_L}, \tag{10}$$

$$\Pi_S^{S,t=2} > \Pi_L^{S,t=2} \text{ if } \alpha < \alpha_{LS} = \frac{2-k\frac{D_H}{2}}{2-kD_H}, \tag{11}$$

$$\Pi_H^{S,t=2} > \Pi_0^{S,t=2} \text{ if } \alpha < \alpha_{0H} = \frac{2-k\frac{D_H}{2}}{2-kD_H}, \tag{12}$$

$$\Pi_S^{S,t=2} > \Pi_0^{S,t=2} \text{ if } \alpha < \alpha_{0S} = \frac{(1-p)\,D_H\left(2-k\frac{D_H}{2}\right) + pD_L\left(2-k\frac{D_L}{2}\right)}{(1-p)\,D_H\,(2-kD_H) + pD_L\,(2-kD_L)}, \tag{13}$$

$$\Pi_H^{S,t=2} > \Pi_L^{S,t=2} \text{ if } \alpha < \alpha_{LH} = \frac{(1-p)\,D_H\left(2-k\frac{D_H}{2}\right) - pD_L\left(2-k\frac{D_L}{2}\right)}{(1-p)\,D_H\,(2-kD_H) - pD_L\,(2-kD_L)}, \tag{14}$$

Note that Eqs. (10) - (12) imply that the profitability to offer a prize for the H-type

[11]Note that we implicitly assume here that with a prize for H-type vulnerabilities S does not acquire the L-type vulnerabilities in $t = 2$. If it would do so, the outcome would be the same as offering two prizes.

(L-type) hacks is independent of whether the bounty program also includes a prize for the L-type (H-type) hacks, i.e., $\alpha_{0H} = \alpha_{LS}$ and $\alpha_{0L} = \alpha_{HS}$. In order to identify the optimal prizes, we characterize the order of the critical values in Lemma 3.

Lemma 3. *It holds that*
$\alpha_{0L} = \alpha_{HS} < \alpha_{0S} < \alpha_{0H} = \alpha_{LS} < \alpha_{LH}$.

Proof. See the Appendix.

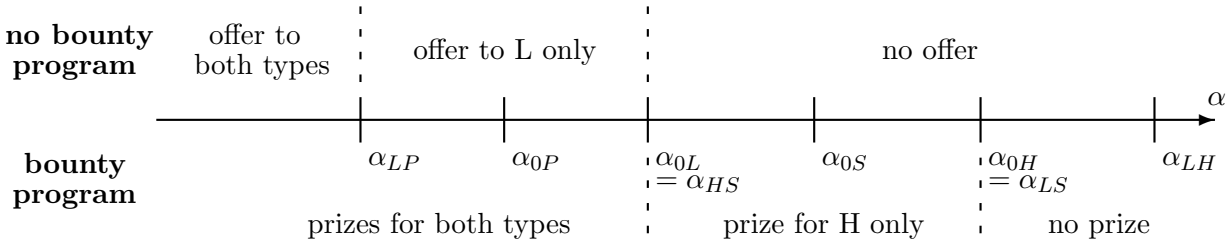

Figure 4: Optimal offers for $p > \tilde{p}$

The lower part of Figure 4 illustrates the consequences of Lemma 3 for the optimal strategy. The upper part shows the optimal offers without a bounty program for comparison. For low levels of α, the software developer offers separate prizes for each type of hack. Once $\alpha \geq \alpha_{H,S} = \alpha_{0,L}$ it becomes too costly to pay for L-type hacks and the software developer prefers the damage caused by exploitation. However, the software developer continuous to offer a prize for H-type hacks. Only after α increases above $\alpha_{H,S}$, paying for H-type hacks becomes too expensive. We summarize this result in Lemma 4.

Lemma 4. *With a bounty program, the software developer*

1. *announces the prize $P_L = \alpha D_L (2 - kD_L)$ for L-type hacks and $P_H = \alpha D_H (2 - kD_H)$ for H-type hacks if $\alpha \leq \alpha_{HS}$. The hacker accepts the prizes.*
2. *announces the prize P_H for H-type hacks and no prize for L-type hacks if $\alpha_{HS} < \alpha \leq \alpha_{0H}$. The hacker accepts the prize.*
3. *announces no prize if $\alpha > \alpha_{0H}$.*

3.3.1. Comparison

The bounty program allows the software developer to perfectly differentiate between the two types of vulnerabilities. By committing to pay the pre-defined prize, the hackers reveal the type of their vulnerability, and the information asymmetry gets resolved. Consequently, the software developer can pay each type of hacker the required amount. This

has two advantages. First, the software developer can pay a lower amount to L-type hacks. Second, he can offer a payment for H-type hacks only. However, the software developer incurs the fixed costs F for maintaining the program.

Comparing the payoff following from Lemma 4 with the payoff in Lemma 2 allows us to state Proposition 1.

Proposition 1. *The incremental payoff of a bounty program is given by*

$$\pi^S = \begin{cases} p\alpha\left(D_H\left(2-kD_H\right)-D_L\left(2-kD_L\right)\right)-F & \text{if } \alpha \leq \alpha_{LP} \\ (1-p)\,D_H\left(\left(1-\alpha\right)\left(2-kD_H\right)+\frac{k}{2}D_H\right)-F & \text{if } \alpha_{LP} < \alpha \leq \alpha_{0H} \\ -F & \text{if } \alpha > \alpha_{0H} \end{cases}$$

The advantages of the bounty program depend on the parameter setting. In the low α case, the bounty program allows the software developer to perfectly differentiate between the two types, rather than paying off both types with the high payment. In the intermediate α case, the bounty program saves the software developer the difference between its potential damages and the market value, e.g., reputation costs, and the resources that are spend on detection for H-type hacks. For a larger α, it would be too costly to pay for any of the two vulnerabilities as the market values outweigh the damage costs.

Hence, the decision whether the software developer employs a bounty system depends on the trade-off between the benefits and costs of it. We define the critical fixed costs F^c such that the incremental payoff of the system just equals zero. Analyzing the comparative statics of F^c, allows us to state Proposition 2.

Proposition 2.

- *In the low α case the critical fixed costs F^c increase in α, D_H and p but decrease in k and D_L.*
- *In the intermediate α case the critical fixed costs F^c increase in k and D_H but decreases in α and p.*
- *In the high α case there is no effect of any parameter on F^c.*

Figure 5 illustrates these comparative static effects. An increase of D_H unambiguously decreases F^c because D_H increases the costs for detection as well as the savings by avoiding excess payments for L-type hacks in the intermediate α case. In the same line of reasoning,

the damage of the L-type gaps does not influence F^c in the low α case because with and without a system, the software developer always pays the same amount for L-type hacks. However, in the low α case, F^c decreases in D_L because a higher low-type damage decreases the benefits of differentiation between the two types. Furthermore, both damages affect the likelihood of the two cases. An increase of D_L and D_H makes the low α case case more likely.

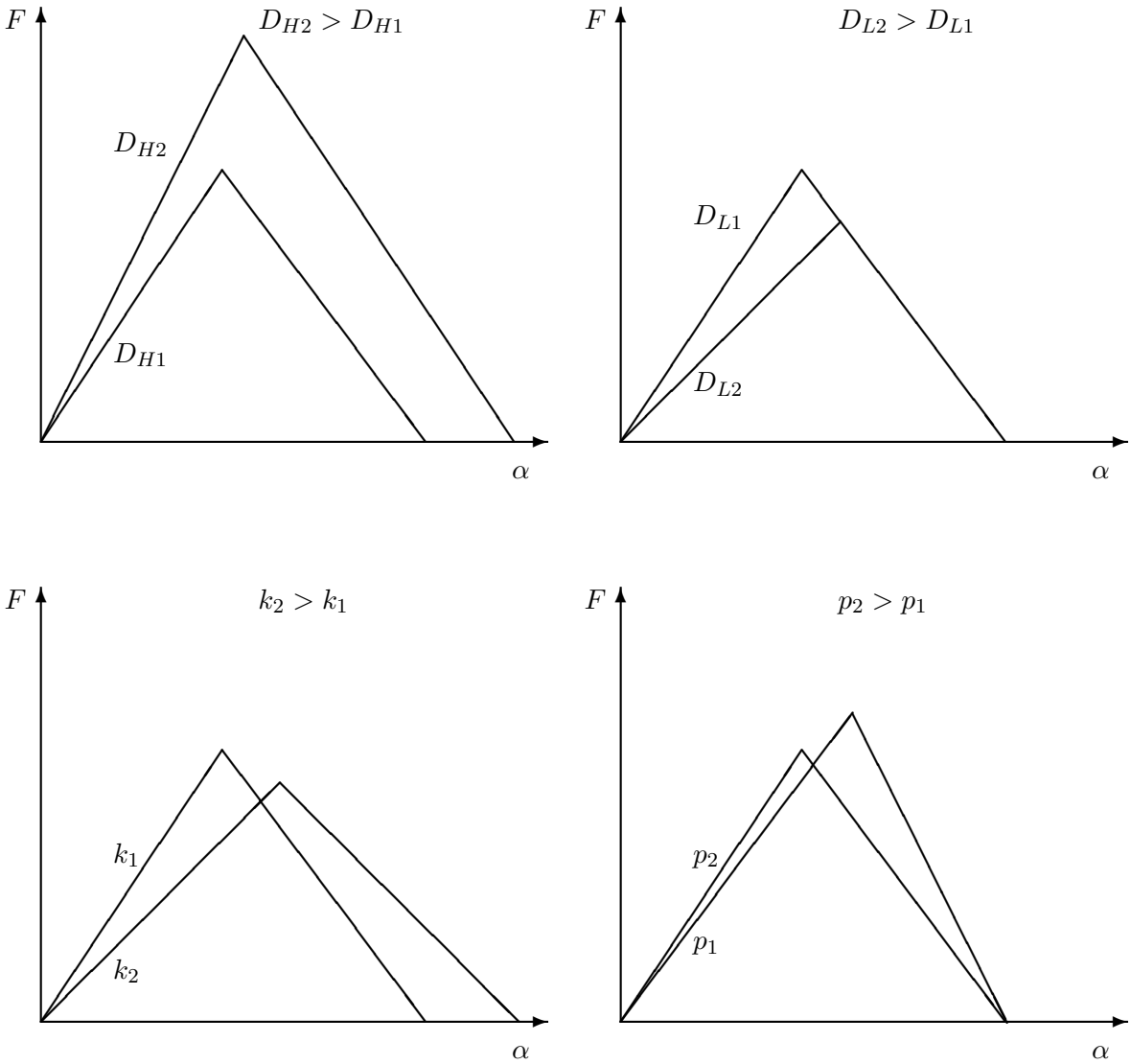

Figure 5: Comparative static effects

Increasing the ex-ante security level (p) decreases the payoff of the bounty program in the intermediate α case because avoiding the reputation and detection costs for H-type hacks becomes less important. Furthermore, the payoff in the low α case increases

because differentiation becomes more important the higher the share of L-type hacks is. Furthermore, an additional payoff decreasing effect stems from the influence of p on the likelihood of the two cases. An increase in p makes the intermediate α case more likely.

Finally, the efficiency of the ex-post detection system affects the payoff of the bounty program, too. The more efficient the system is (higher k), the less beneficial is the bounty system in the low α case. This is because a more efficient ex-post detection system makes an ex-ante differentiation of the two types less beneficial. However, the contrary is true in the intermediate α case case. A higher k lets the software developer spend more on detection, increasing his absolute detection costs. These costs are avoided with a bounty program for the H-type hacks in the intermediate α case.

4. Discussion

By offering a bug bounty program software developers commit to pay for a vulnerability. This commitment allows the software developer to differentiate between the types of vulnerabilities, and the software developer can either offer individual prizes to the different types of vulnerabilities, or offer only prizes to a subset (e.g., only vulnerabilities of type H). The differentiation leads to lower costs and damages for the software developer. However, the benefits of a bug bounty program depend on the specific environment. In general, we find that a bounty program becomes more beneficial the higher the maximal harm from an exploit is (higher D_H). For other characteristics our analysis paints a more nuanced picture.

In principle, our model captures three possibilities for software developers to improve protection against being exploited. First, better ex-ante code surveillance (higher p). Second, an efficient ex-post detection system (higher k), and, third, a bug bounty program. Our analysis reveals that these three alternatives crucially interact.

For software developers prone to particularly pronounced reputation costs relative to the market value of the exploit (small α), a bounty program is more beneficial the better a software is checked ex-ante (higher p). The ex-ante quality and a bounty program complement each other because the more secure the system is ex-ante, the more the

developer can profit from identifying the few severe vulnerabilities, instead of paying off all types with a relatively high amount. In contrast, having an efficient ex-post detection system decreases the profitability of a bounty program. A bounty system and an ex-post detection system are substitutes.

For software developers that face less pronounced reputation costs relative to the value of an exploit (intermediate α), this observations switches. In that case, better ex-ante code surveillance substitutes for a bounty program. The reason for that result is that the fewer high damage vulnerabilities exist, the less important it becomes to maintain a system that identifies them; it is cheaper to incur the reputation costs in that rare case. In contrast, a bounty program benefits from a strong ex-post detection system, i.e., both complement each other. This is because a better ex-post detection is more costly in absolute terms, and the bug bounty program helps avoiding those costs.

Clearly, if the reputation costs are significantly lower than the market value of a vulnerability (high α), a developer would never pay for the information, and rather accept exploitation. Hence, a bounty program does not offer any benefit in that case. Examples for such cases are vulnerabilities that are used by secret services. Even though such an exploit may involve a significant reputation cost, the market value is so high that the developer would not be willing to acquire the information on the vulnerability.

In sum, our results imply that software developers should follow an holistic approach regarding the security of their products. One can easily imagine differences in the reputation costs depending on the type of software product. For example, software developers that deal with sensitive business information are likely to be highly negatively affected by the negative publicity of an exploit. The same is likely to be true for developers that deal with financial transaction, e.g., online banking. If these type of developers decide to implement a bounty program, they should focus their other efforts on better ex-ante coding rather then investing in an efficient ex-ante detection program. The contrary is true for companies that are less affected by reputation costs, or for developers that deal with vulnerabilities that have a relative high market value compared to the reputation

costs.

5. Conclusion

We analyze the benefits of bug bounty programs for software developers. We build a model that is micro-founded in the specific characteristics of the market for vulnerabilities. We find that the benefits of bounty program do not only depend on the characteristics of the underlying software but also that a bounty program crucially interacts with other potential elements of the security strategy.

References

Apple, 2019. Apple Security Bounty. `https://developer.apple.com/security-bounty/`. Accessed: 2020-05-04.

Arora, A., Telang, R., Xu, H., 2008. Optimal Policy for Software Vulnerability Disclosure. Management Science 54, 642–656.

August, T., Dao, D., Kim, K., 2019. Market segmentation and software security: Pricing patching rights. Management Science 65, 4575–4597.

August, T., Tunca, T., 2011. Who Should Be Responsible for Software Security? A Comparative Analysis of Liability Policies in Network Environments. Management Science 57, 934–959.

Choi, J.P., Fershtman, C., Gandal, N., 2010. Network security: Vulnerabilities and disclosure policy. The Journal of Industrial Economics 58, 868–894.

Cooper, C., 2018. WannaCry: Lessons Learned 1 Year Later. `https://symantec-enterprise-blogs.security.com/blogs/feature-stories/wannacry-lessons-learned-1-year-later?` Accessed: 2020-05-04.

Finifter, M., Akhawe, D., Wagner, D., 2013. An empirical study of vulnerability rewards programs, in: Proceedings of the 22nd USENIX Security Symposium, pp. 273–288.

Goodin, D., 2016. iPhone xploit bounty surges to an eye-popping $1.5 million. https://arstechnica.com/information-technology/2016/09/1-5-million-bounty-for-iphone-exploits-is-sure-to-bolster-supply-of-0days/. Accessed: 2020-05-04.

Kelley, G., 2015. Windows 10 Upgrades Cannot Be Stopped. https://www.forbes.com/sites/gordonkelly/2015/06/26/free-windows-10-upgrades-danger/#63912ea75962. Accessed: 2020-05-04.

Telang, R., Wattal, S., 2007. An empirical analysis of the impact of software vulnerability announcements on firm stock price. IEEE Transactions on Software Engineering 33, 544–557.

Appendix

Proof of Lemma 1

1. $\alpha_{0L} > \alpha_{0P}$ iff

$$D_H(2-kD_H)\left(2-k\frac{D_L}{2}\right) > (1-p)D_H\left(2-k\frac{D_H}{2}\right)(2-kD_L) + pD_L(2-kD_L)\left(2-k\frac{D_L}{2}\right)$$

$$D_H(2-kD_H)\alpha_{0L} > (1-p)D_H\left(2-k\frac{D_H}{2}\right) + pD_L\left(2-k\frac{D_L}{2}\right)$$

$$D_H(2-kD_H)\alpha_{0L} - D_L\left(2-k\frac{D_L}{2}\right) > (1-p)D_H\left(2-k\frac{D_H}{2}\right) - (1-p)D_L\left(2-k\frac{D_L}{2}\right)$$

$$\Delta_1\alpha_{0L} > (1-p)\Delta_2$$

$$\Rightarrow p > \tilde{p}$$

2. $\alpha_{0P} > \alpha_{LP}$ iff

$$\left((1-p)D_H\left(2-k\frac{D_H}{2}\right) + pD_L\left(2-k\frac{D_L}{2}\right)\right)(D_H(2-kD_H) - pD_L(2-kD_L))$$
$$> (1-p)D_H^2(2-kD_H)\left(2-k\frac{D_H}{2}\right)$$

$$\left(2-k\frac{D_L}{2}\right)(D_H(2-kD_H) - pD_L(2-kD_L)) - (1-p)D_H\left(2-k\frac{D_H}{2}\right)(2-kD_L) > 0$$

$$\alpha_{0L}(D_H(2-kD_H) - pD_L(2-kD_L)) > (1-p)D_H\left(2-k\frac{D_H}{2}\right)$$

$$\alpha_{0L}(\Delta_1 - (p-1)D_L(2-kD_L)) > (1-p)D_H\left(2-k\frac{D_H}{2}\right)$$

$$\alpha_{0L}\Delta_1 > (1-p)\Delta_2$$

$$\Rightarrow p > \tilde{p}$$

Proof of Lemma 3

1. A comparison of α_{0H} and α_{0L} shows that $\alpha_{0H} > \alpha_{0L}$

2. $\alpha_{0L} > \alpha_{0S} > \alpha_{0L}$

   a) For $p = 1$, it follows that $\alpha_{0S} = \alpha_{0L}$.

   b) For $p = 0$, it follows that $\alpha_{0S} = \alpha_{0H}$.

   c) Because $\frac{\partial\alpha_{0S}}{\partial p} = \frac{D_H D_L k(D_H - D_L)}{((1-p)D_H(2-kD_H)+pD_L(2-kD_L))^2} < 0$, a) and b) imply $\alpha_{0H} > \alpha_{0S} > \alpha_{0L}$.

3. $\alpha_{LH} > \alpha_{0H}$

   a) For $p = 1$, it follows that $\alpha_{LH} = \alpha_{0H}$.

   b) Because $\frac{\partial\alpha_{LH}}{\partial p} = \frac{-D_H D_L k(D_H - D_L)}{((1-p)D_H(2-kD_H)-pD_L(2-kD_L))^2} > 0$, a) implies that $\alpha_{LH} > \alpha_{0H}$ for $p > \tilde{p}$.

NORGES HANDELSHØYSKOLE
Norwegian School of Economics

Helleveien 30
NO-5045 Bergen
Norway

T +47 55 95 90 00
E nhh.postmottak@nhh.no
W www.nhh.no